Development and command-control tools for many-robot systems

Douglas W. Gage

RDT&E Division, Naval Command Control and Ocean Surveillance Center
NCCOSC RDTE DIV 531, San Diego, CA 92152-7383

ABSTRACT

An initial concept is presented for a set of communications and command-control capabilities that can facilitate the development process for a system consisting of an arbitrarily large number of relatively simple (and probably small and inexpensive, although not necessarily "micro") robots -- mechanisms to allow the human developer to quickly and easily see into the internal state of large numbers of robots, and to quickly and easily make changes to the robots' behaviors. The interface between the control station and the developer must be carefully designed in order to provide a serviceable development environment, and this environment, or subsets of it, should then evolve into the operator's station for deployed systems that require active control and monitoring. The development environment must provide the developer with a precise model for (re-) programming the elements of the system, while the system operator will require only the simplest functional model of the system as a whole that can support mission needs, combined with a convenient way to tell the system what to do.

1. INTRODUCTION

Eight years have now passed since Anita Flynn described a vision of swarms of "gnat robots" performing useful tasks for humankind [1], and they're not here yet. While MEMS (Micro Electro Mechanical Systems) as a manufacturing technology has moved steadily forward, the principal commercially successful applications for MEMS devices have been components such as pressure sensors and accelerometers for the automobile market. In fact, robots of any size have yet to appear in our daily lives -- arguably the closest thing to a recognizably-robotic sensor-actuator combination we see is the proximity-sensing automatic door opener, together with its descendants, the toilet flushers and faucet controls now seen in many airport restrooms.

Nevertheless, robotics researchers persist, and there is little doubt that the continuing exponential improvement of microelectronic processing price/performance, coupled with continuing developments in MEMS and other sensor and actuator technologies, will eventually yield successfully mass-marketed autonomous mobile robots -- for example, an early possibility might be toy "pets" capable of displaying interesting behaviors in their interactions with their owners and with each other. Even currently available toy cars can be (and often are) easily coupled with fairly simple electronic sensor/processor/control appliques to provide affordable robotic research platforms.

The mass production of robots will certainly trigger dramatic unit-price reductions, and it is these reduced costs that will finally permit the implementation of "swarms" of "mini" (and, ultimately, "micro") robots to handle real-world applications in both the military and civilian worlds. But the realization of practical systems comprising large numbers of mobile robotic elements which are capable of performing useful tasks requires much more than just the cost-effective manufacture of the robots themselves -- for example, the prospective user must understand what the system is capable of doing in order to know when to deploy it, must know how to tell the system to do the specific task required, and must be able to assess how well the system is doing or has done its job. And the system development process must ensure that the system will actually achieve its advertised level of performance across the full range of specified manufacturing tolerances and intended operational environments

In previous papers we have presented (a) the notion of “coverage” as a paradigm for the system level functionality of many robot systems [2], (b) some initial notions of sensor-based behaviors to implement various coverage modes [2,3], (c) measures of effectiveness and system design considerations for the generic area search application (e.g., minesweeping) [3,4], (d) an approach to the communications needs of a system consisting of an arbitrarily large number of simple autonomous robots [5], and (e) an examination of some of the factors that will determine how we can most successfully develop practical and cost-effective many-robot systems [6]. In this paper we propose a set of tools to help the developer of a many-robot system through the often agonizing process of testing and refining ("debugging") the system hardware and software -- tools which can also be adapted to provide necessary control and monitoring capabilities to the user of a deployed many-robot system.

2. THE PROBLEM

A specific "nightmare" scenario motivates this paper:

You are the lead engineer developing a many-robot system intended to address a real-world application -- for example, to find and destroy buried antipersonnel landmines in fairly benign terrain. The plan is to add sensors and processing to inexpensive "toy car" mobility bases so that they will move as a churning flock whose center wanders in a quasi-diffusing random motion from its deployment point. The system concept calls for a robot to stop when it finds a mine and then (in its production version) blow up both itself and the mine at a predetermined time, or sooner if it is disturbed. The system is intended for deployment behind enemy lines in advance of an attack, and therefore is designed to require absolutely no communication. The system is designed to be completely scalable -- the more elements are deployed, the larger the flock size and the larger the area covered by the diffusion. The project's (very reasonable) development strategy called for breadboarding up a dozen or two individual robots, experimentally determining sensor characteristics and performance, and designing a set of reactive behaviors. Following validation of the behavior algorithms with extensive simulation, and some real product engineering, several hundred elements have been manufactured for an initial capabilities demonstration.

It's demo day, and you, as project lead engineer, are standing with your program manager and sponsor at the edge of the demo field when the box of critters is dumped from the back of a HMMWV. They come streaming out, and start to form the desired flock. As everyone starts congratulating each other, your sponsor suddenly motions towards a small but growing clump of robots off to one side and says "Hey! What's that little group doing over there?" As you listen to your project manager promising your sponsor that you will obviously be able to quickly fix this very small glitch in the system, you realize that not only do you not have a way to reprogram the units in the field, but also that you wouldn't know how to reprogram them if you did, since you don't have a clue as to what is causing the aberrant behavior, which did NOT appear in simulation, or in your preliminary tests at a field that appeared to be virtually identical to the demo site. Furthermore, you don't know how you are going to find out what is going wrong so that you can fix it.

While this scenario in its details has admittedly been deliberately constructed as a worst case, the problems it poses are all too real. An autonomous mobile robot is a complex machine which is called upon to perform navigational and other tasks framed by its human developers, but without being able to call upon human perceptual capabilities. The process of debugging such a machine is often extremely frustrating, because the precise features of the environment that the robot's inexpensive sensor suite responds to may not be those that the developer intended, so that a scheme that works well in one environment may fail miserably in another environment which appears imperceptibly different to the human developer [7]. The developer's problem is exacerbated by the fact that the robot's externally observable behaviors may offer few clues to the dynamics (sensor inputs and processing states) producing them. What looks to the observer like a nice straight-line robot path may "feel" to the robot like a rapid oscillation between small left and right turns at a tens-of-Hertz rate (it is a control loop, of course). Since the notion of an "operator's interface" for a robot that is supposed to be completely autonomous sounds like an oxymoron, such a capability is often implemented only as a rudimentary afterthought -- but it is precisely what is needed for efficient debugging.

When we move from a single robot to a system consisting of many robots, the problem is multiplied manyfold. The cost of each element must be minimized to keep the system competitive, which means that we can absolutely expect to encounter the sorts of problems that result from limited sensor capabilities. Desired ensemble behaviors may be intended to "emerge" from the simpler behaviors of the individual elements, adding an additional layer of complexity. The system concept of operations may involve little or no communications when deployed, either between elements or between the user (the person who deploys the system) and the elements. Finally, the system may be intended to be used in a single shot "throwaway" mode, so that the target hardware is neither reprogrammable nor rechargable.

The goal of this paper is to propose some generic tools that the developer can use to facilitate the debugging process for systems comprised of an arbitrarily large number of autonomous robots.

3. THE "SLEEP" COMMAND CHANNEL

The key element of this scheme is a conventional relatively high bandwidth broadcast channel which is employed to transmit messages from the user's workstation to the robots. Appropriate error detection coding is applied to the messages, with forward error correction also desired since the arbitrarily large number of robots precludes the general use of negative acknowledgements (NAKs) as well as of positive ACKs. Security measures such as encryption are implemented as appropriate to the application, and, while critically important in many applications [8], are not considered further in this paper. How this command channel is implemented in hardware (and how "high" a bandwidth it provides) will be extremely application dependent, although RF and optic schemes certainly come quickly to mind, and also is not considered further. It is certainly possible to envision alternate schemes using non-broadcast channels and message relaying, but these would involve considerations of routing and complicated system state, and might permit a failure in one node to prevent (or at least delay) the user from communicating with a desired target node.

The first question is how to address a message to a specific individual robot or group of robots, as the number of robots grows arbitrarily large. Since the robots are identical and interchangeable (at least within each of a relatively small number of castes), the primary mode of addressing for the system could be termed "dynamic situated group addressing" -- the equivalent of saying "anyone within 50 meters of the wall who has more than 50% fuel level and at least one grenade, you're now in Group Alfa", and then addressing future messages directly to "Group Alfa." Essentially, each robot determines whether a given command is intended for him by evaluating a predicate expression based on the values of his own state variables.

This rule-like paradigm -- "IF <predicate> is true, THEN store and execute <command>" -- has been further evolved and simplified by observing that, if we adopt a LISP representation for <predicate>, we might just as well also use it for <command>, and in that case the IF-THEN rule functionality can be explicitly captured in a LISP COND function, and the whole message becomes a single LISP expression whose evaluation is all the processing that is required. And as long as our node is going to have to be able to evaluate LISP expressions received as messages, why not incorporate a writeable program store (rule base) of LISP expressions to use for debugging and even for high level mission management? The resulting scheme is nicknamed "SLEEP", for "Simplified LISP-like Expression Evaluation Paradigm", and is described in some detail in section 8 below, following a presentation of additional capabilities of interest.

4. REPLIES ON THE COMMAND CHANNEL

Even with an arbitrarily large number of robots sharing a single broadcast channel, it is possible for the user to initiate a one-to-one session with one of a situationally addressed group of nodes, no matter what their number, and the same process will tell the user approximately how many members actually are in the group. The key is to understand and exploit the statistics of a contention communication channel, and to command the nodes to transmit a message with a probability chosen so that one of the messages will get through in short order, where the definition of "short order" will depend on the characteristics of the command channel's physical layer.

For example, on a carrier-sense multiple access (CSMA) channel like Ethernet, a node desiring to transmit a message can defer its own transmission when it detects another transmitter's signal carrier on the channel. Collision-detection (CD -- another Ethernet property) permits a transmitting node to immediately determine if its own transmission is colliding with that of another node, and it is therefore able to "jam" the channel to guarantee that all transmitting nodes detect the collision and then "backoff" according to some prescribed algorithm before transmitting again. Since CSMA and CD capabilities can only help make channel utilization more efficient, let us consider the worst possible case of a pure contention channel, in which each node must transmit "blind". This is the case, for example, in a satellite channel, and the analyses of the *pure Aloha* and *slotted Aloha* access techniques developed for satellites in the early 1970s apply [9].

At any point in time, the number of nodes transmitting is either 0, 1, or greater than 1. If exactly one node transmits for the entire duration of its message, then the message is received by the other nodes; if more than one node transmits during any portion of a message, then the messages "collide" and are not received. If nodes are permitted to start to transmit whenever they desire (pure Aloha), then a maximum throughput efficiency of about 18% is achievable when the offered load (number of nodes times probability of each node transmitting at any point in time) is equal to 0.5. If the nodes are constrained to start their message transmissions at specific points in time (i.e., time is divided into *slots*), then the maximum efficiency is about 36%, occuring when the offered load is 1.0.

So the master simply makes an estimate E of the number of nodes in the group, and commands them all to transmit, with probability p = 1/E, a brief message into each of a specified small number of slots (2-10). If E is significantly greater than the actual number of nodes N, then the actual offered load will be too low and the slots will all be empty. If E is too small, then all the slots will see collisions (see Figure 1 below). In either case, no messages will be received by anyone. If the master node is able to distinguish between an emply slot and a collision (it doesn't matter if the slave nodes can do so), then it is possible to perform a binary search in log E. If empty slots and collisions look the same to the master (which is entirely likely with a covert channel), then the master must scan the entire range of possible N until he receives a message -- a simple geometric scan of E with the value of E changing by a factor of 2 for each successive slot has a probability of approximately .81 of yielding at least one successful message. Providing two slots for each value of E thus will yield better than .96, and three slots achieves .993.

In either case, once E is close to N, additional probes using larger numbers of slots can determine N to whatever precision is desired, and the master can send messages to the individual unique addresses returned in the successful response messages.

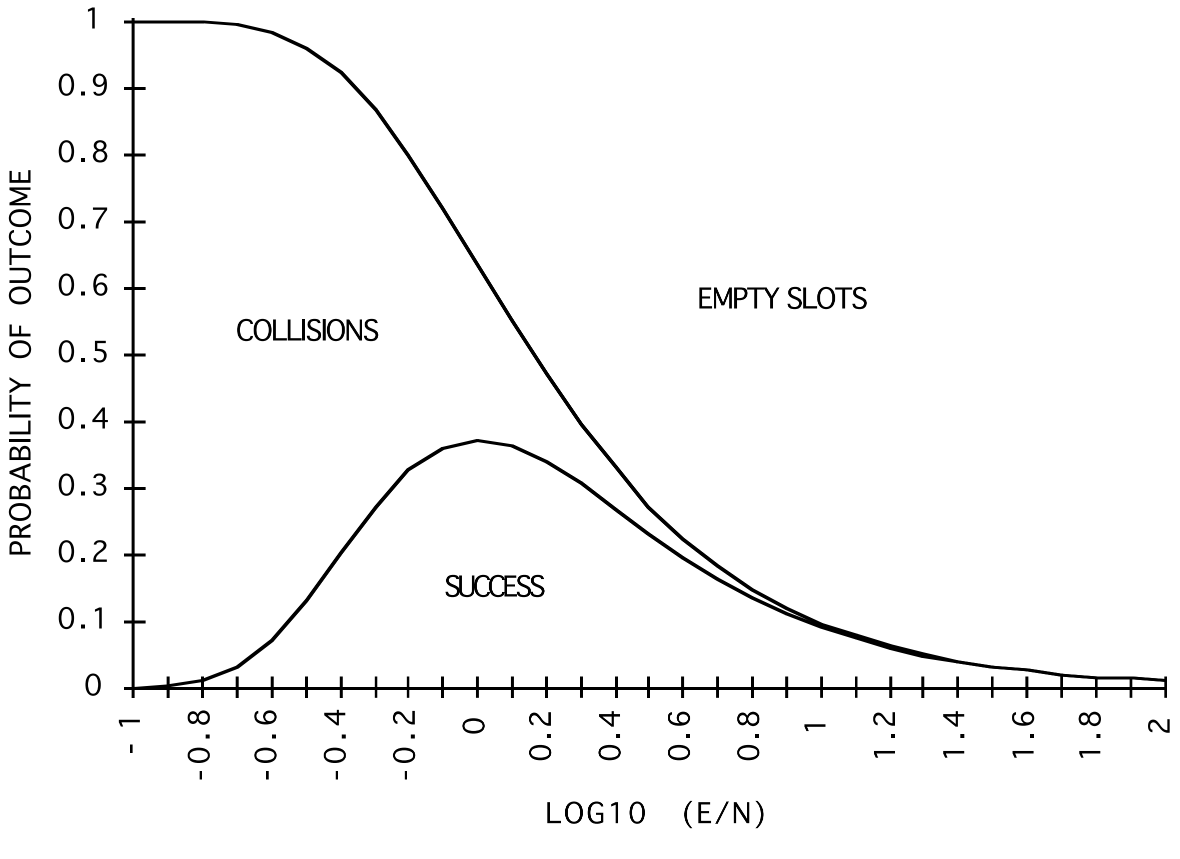

Figure 1. Probabilities of successful message transmission, collisions, and empty slots for Slotted Aloha channel access, plotted as a function of the logarithm of the ratio of estimated number of nodes to actual number. Log (E/N) = 0 corresponds to an offered load of 1.0. Empty slots occur to the right of the peak, collisions to the left.

The systems of "gnat" sized and (potentially much) smaller robots that MEMS technology may ultimately make feasible will involve additional communications difficulties. Severe power limitations will probably preclude the use of actively powered transmitters over any significant distance, including the display of lights proposed below. One option (which has been proposed several times in the past, at least as early as 1977) is to incorporate a modulated retroreflector into the "slave" units. The active node (master) illuminates the passive nodes with a laser beam, and the passive nodes modulate the signal which is retroreflected back to the active node. The carrier beam can be directly modulated for half-duplex operation, or two modulated subcarriers can be employed to provide full-duplex communications. This scheme allows the active node (the master) to provide all the power for communications in both directions. In addition, the passive nodes do not require a precise pointing system to achieve this high signal gain. The development of a "standard cell" MEMS modulated retroreflector would be a valuable addition to the microrobot toolbox. Pister and his students at UCLA have explored the fabrication of such devices. [10]

5. DISPLAY "LEDS"

In order to make effective use of internal state information acquired from a specific robot, this data must be correlated with external observations of the robot's behavior. This will often require identifying precisely which one of the many identical robots is the one doing the reporting. This capability is just one of many provided by a low bandwidth cueing and display mechanism incorporated into each robot -- for small ground vehicles in a laboratory environment it's convenient to think of this as a set of several different colored LEDs, although the implementation modality may be markedly different for other systems in other environments. The master can identify a single robot -- or any group of robots -- by instructing them to "wink" their yellow LEDs.

This same mechanism can be used to explicitly display state information to the user, or to share information with its neighboring elements. Via the broadcast command channel, the master assigns -- and can dynamically reassign -- a specific representational meaning to each LED or combination of LEDs; coded modulation of the LEDs is not used to send serial messages. The system developer can program a light to indicate a specific boolean combination or sequence of behavior states in order to help understand an unexpected group behavior by, for example, "capturing" an unanticipated transient state. In a deployed system, each element may inform its neighbors of its own behavior state in order to more effectively coordinate the behavior of the local group (Wang [11] refers to this as "signboard" communication, and similar schemes have been explored in simulation by Arkin et al [12] and Lucarini et al [13]). When a low cost "early warning" sentry robot detects an intruder, it can signal that fact to attract the attention of a more capable and expensive "assessment and response" robot (analogous to the action of antibodies and phages in the immune system).

During development, one light is used for error signalling. A hardware timer turns the red LED on if the robot's software fails to send its periodic timer reset. The software can also be programmed to turn the same LED on (continuously or flashing) if it fails to receive the "keep-alive" message periodically sent by the master, thus providing the highly desirable "please tell me if you don't receive this message" function, and allowing the culling of failed units. The LEDs are memory mapped, with the additional timing hardware for the error LED being transparent to the programmer (who can of course read the value of the variable to determine if the timer has in fact timed out and turned on the LED).

A key benefit of using a display mechanism such as colored lights which can be directly imaged with a camera is the potential for automatic aggregation and display of status and reporting information at the master. Instead of the master having to process an endless torrent of digital messages containing state information, maintaining a database containing each element's position and status, and repeatedly updating a display, the master can, if the geometry of the situation permits, simply program a rapid sequence of display light assignments and use simple image processing to derive the desired information. The human user of a sentry system deployed in and around around a storage yard could use a camera mounted on a mast or aerostat to visually see where the robots are and what they "think" is happening. On a smaller scale, the elements of a distributed instrumentation system could be glued to the side of a piece of equipment and a camera set up to provide a continuous display of thermal, acoustic, or other parameters. And microsopic sized sensing elements programmed to change their optical characteristics in response to their chemical and/or physical environment could be suspended in a container of liquid to provide "in-situ visualization" of temperature, pH, or salinity.

6. "MARKING" ROBOTS

A valuable complement to the display LED mechanism is a designation mechanism which allows the user to gain the attention of a specific unit by physically pointing to it -- the equivalent of "Hey you.... yes, you!". The user has a device that looks like a flashlight that emits both visible light (so the user can see where he is pointing it) and an IR beam that can be switched between two simple modulation modes named *mark* and *unmark* (e.g., a square wave at two different frequencies). The control station sends a command to a group of nodes to enter the *marking* state. While *marking*, hardware in a node sets its *marked* variable whenever it detects the *mark* beam, and clears *marked* when it detects *unmark*. By viewing an LED programmed to track the *marked* variable, the user can precisely designate the units he wants, then clear *marking* and address messages to the *marked* subset. A physical button that toggles the *mark* variable might be a useful feature on "toy car" class robots.

7. ACTIVATING AND "SUSPENDING" ROBOTS

The task of activating a system comprised of a potentially unlimited number of robots is not trivial, especially since a deployed system will probably have to have significant "shelf life" -- think in terms of a carton of heavy packing peanuts stored in a locker somewhere. Underwater robots can be activated by exposure to seawater; in some other situations exposure to ambient light could be used. But the general solution calls for some (probably application-specific) stimulus that can be detected by a bit of hardware that consumes zero power while waiting. This stimulus can be used to turn on the node's core processor, which can then either process more complex corroborating stimuli, or proceed directly to initializing the entire robot and awaiting (if applicable) a command from the control station.

The task of node activation must also be accommodated in the system development process. Since the developer will often prefer to turn nodes on and off without having to perform physical rituals such as pouring water on them, the command structure should provide message types to support this. A node in *sleeping* state plays possum while waiting for a WAKE command. A *frozen* state permits the developer to halt the node's execution of its mission program to provide an opportunity to examine its internal state.

8. SLEEP OPERATION, FUNCTIONS, AND VARIABLES

Basically, SLEEP consists of several address spaces, with associated storage classes and processing: (1) expressions, (2) variables, and (3) functions.

Each node incorporates an indexed set of buffers for storing a number of LISP-syntax expressions. For convenient reference in programming, the operator can associate a name with a specific buffer at his workstation. Each expression buffer is either *active* or *inactive*, and this can be changed under program control, either (a matter of implementation detail) by SETQing the active flag variable or through an explicit *activate* or *deactivate* function call. The core SLEEP process merely evaluates the expressions in the active buffers in round-robin sequence (although any of a number of different priority schemes could certainly be added). The message reception process writes an incoming expression into a special buffer and SLEEP executes it one time (the expression can, of course, copy a portion of itself into one of the other regular buffers for repeated processing as part of the active ruleset).

SLEEP provides an indexed set of variables, which are used in three ways. Some point to the variables used by the robot's underlying processing (including memory-mapped sensor inputs and actuator outputs), so that the user can have maximum flexibility in reading the unit's state. Others support SLEEP processing itself, like *marking*, *marked*, *frozen*, and *sleeping*. The user has complete discretion in using the rest of the variables, associating names with them and using them in expressions.

Functions include basic arithmetic (+, -, *, /, ^, etc.), boolean (AND, OR, NOT), LISP "special forms" [14] (COND, SETQ), SLEEP support (power_down), and pointers to functions in the robot's core software. Randomization functions include (PROB x), which evaluates as TRUE with probability x, and the special form (DICE (x A)(y B)(z C)), which evaluates A with probability x, B with probabilty Y, and C with probability z.

In the initial (very preliminary) exploratory C-based implementation of SLEEP, the range of LISP-expressions has been simplified considerably, and, in fact, expressions are not even represented as conventional linked lists. Instead, an expression, whether in a message or in the rule base, is represented as a sequence of tokens. Each token represents a number, a variable, a function, or an expression in the rule base. So far, numbers have been limited to integers only, but it is clear that floating point numbers will be absolutely required because of the necessity of representing probabilities. The initial set of functions implemented include arithmetic and logical, plus the LISP special forms [14] COND, SETQ, and a variant of SETQ called SETQQ which QUOTEs both its arguments. Moreover, an expression always evaluates to a number, which can, of course, also serve as a boolean. So this is not really LISP, but it captures some relevant desirable features of LISP such as call-by-value parameters, operator prefix notation, and function side effects.

9. AN ILLUSTRATIVE APPLICATION

To illustrate the role that SLEEP might play in a system comprised of large numbers of simple behavior-based robots, let us consider how it might be applied to Hoskins' "emergent Braitenberg Vehicle" [15].

Hoskins' system is comprised of dozens of physically identical mobile vehicles, operating in a plane. All vehicles travel at the same constant speed in straight line paths, each occasionally making a "tumble" -- a turn of about 67 degrees, to the left or right at random. Each vehicle can generate a "ping" which encodes a few bits, and can detect the pings of all the other vehicles. The ping detector can determine the approximate range to the pinger, and just one bit of bearing information -- i.e, whether the ping's origin is in front of it or behind it. A second sensor provides approximate range and and one-bit front/behind bearing to a light source located in the plane. Hoskins divides 50 of these units into 5 *castes* of 10 units each, encodes the pinger's caste number in each ping, and configures each caste with a set of simple behavioral rules so that the group as a whole manifests the key characteristics of a classic Braitenberg Vehicle [16] homing toward the light source, including body structure, propulsion, control, and sensing.

SLEEP is well matched to the behavioral rule structure developed by Hoskins. Lower level "core" software is written to implement the following programming model. A ping is represented as variables *pRange*, *pBearing* (FRONT or REAR), *pCaste*, and *pFlag* (needed for synchronizing the processing of pings as discrete events). The lower level software queues up multiple pings and presents these events one at a time, setting *pFlag*. The SLEEP ruleset resets *pFlag* when it has completed processing the event by making one complete pass through the rules. Similarly, the light source is represented by *sRange*, *sBearing*, and *sFlag*. A *timer* is also necessary to support Poisson processes. The only effector functions are *Ping* with a numerical argument which will always be set to the value of the variable *myCaste*, and *Tumble*, which requires no argument. A typical rule (Hoskins' behavior $b_{4,5}$) would look like:

(COND ((AND pFlag (EQ pCaste 3) (EQ bBearing FRONT) sFlag (EQ sBearing FRONT) (PROB 0.9)) (PING myCaste))

Receiving elements of caste 4 would store this rule in expression buffer 5 and set its active flag when they received it embedded in:

(COND ((EQ myCaste 4)(SETQ expr5 (QUOTE the-expression-to-be-stored)) (SETQ active5 TRUE)))

What is important is not the details of the LISP syntax; rather it is the fact that the processing of messages broadcast to all nodes is handled by exactly the same simple interpretive execution mechanism that processes the rule base as a whole.

SLEEP becomes especially relevant when we visualize the operations of this system not in simulation but in the laboratory. The robots spend their nights at their electrical "feeding trough" recharging station. To perform an experiment, the researcher activates the robots by turning off the recharging station. The units awake, and their core SLEEP software fires up. The red LED marks any units that wake up sick. The control station starts to transmit keep-alive messages at 10 second intervals, and any units that do not hear the control station turn on their red LEDs in a flashing mode, then turn themselves off after a minute or two. The user initiates downloading of the programs for today's experiment. Different experiments may, of course, require not only different behavioral rule-sets, but also different numbers of castes of robots. The program tells the robots to probabilistically assign themselves to one of the castes, then probes to ensure that each caste is approximately equally represented. (Of course, with a total of only 50 or so elements, it is possible to determine the status of every robot, but SLEEP is designed to work no matter how many there are.) Then the behavior rule sets are downloaded into the units' expression buffers. The user can then command the units to assume the desired initial spatial configuration (e.g., randomly throughout the space). This transit can be made using behaviors and sensors that the experiment itself may not utilize (the issue of collisions, for example, must be dealt with). During the experiment, the positions of the robots are captured by a camera mounted on the ceiling, perhaps bandpass filtered for the desired LED. The units can be reconfigured to perform several experiments in succession, and then, following the day's work (or when power starts to run low) be commanded to return to the recharging station.

10. CONCLUSIONS: NODE BEHAVIOR, GROUP BEHAVIOR, AND COMMAND-CONTROL

The approach to many-robot systems described in this paper is almost completely orthogonal to the major threads of research currently being pursued in multi-robot / multi-agent systems, such as emergent behaviors, artificial life (ALife), distributed aritificial intelligence (DAI), and communication and cooperation between peer agents (see Mataric's excellent overview of work in the field [17]). But it is relevant and complementary to these major threads of work -- and on more than one level.

First, on a low level, it offers an approach to the daily problems encountered by researchers working with real robotic hardware, rather than in simulation. As humans, we are so at home in the world created by our vision-oriented perceptual capabilities that we mistake it for the actual physical world around us, and we are therefore constantly suprised and disappointed by the comparatively pitiful capabilities of our robots' sensor subsystems. The researcher needs to be able to "get inside the head" of his or her robots, and this becomes ever more important in systems which attempt to transcend the minimal functionality and performance of inexpensive elements by employing large numbers of them. And tools that make the researcher's job easier also make research dollars go farther.

On a higher level, it can provide command control capabilities which useful many-robot systems will require in addition to group behaviors. Biology provides powerful models of many-agent systems in the social insects, but these have evolved through natural selection, while the "evolution" of many-robot systems will essentially by driven by such systems issues as marketability and manufacturability as well as functionality and designability [6]. To be viable, bottom-up driven "emergent" behavior must respond to the top-down driven requirements of the user who pays for the system, and this often implies at least some minimal level of command control.

11. ACKNOWLEDGEMENTS

The author wishes to thank Dr. Douglas Hoskins for stimulating technical discussions, as well as for the use of his work as an example application domain. This work is supported by the Intelligent Systems Program of the Office of Naval Research, Computer Science Division.

12. REFERENCES

1. Flynn, A.M. "Gnat Robots (And How They Will Change Robotics)", **Proceedings of the IEEE Micro Robots and Teleoperators Workshop**, Hyannis MA, 9-11 November 1987. Also appeared in **AI Expert**, December 1987, p 34 et seq.

2. Gage, D.W. "Command Control for Many-Robot Systems", **AUVS-92, the Nineteenth Annual AUVS Technical Symposium**, Huntsville AL, 22-24 June 1992. Reprinted in **Unmanned Systems** Magazine, Fall 1992, Volume 10, Number 4, pp 28-34.

3. Gage, D.W. "Randomized Search Strategies with Imperfect Sensors", **Proceedings of SPIE Mobile Robots VIII**, Boston MA, 9-10 September 1993, Volume 2058, 270-279.

4. Gage, D.W. "Sensor Abstractions to Support Many-Robot Systems", **Proceedings of SPIE Mobile Robots VII**, Boston MA, 18-20 November 1992, Volume 1831, pp 235-246.

5. Gage, D.W. "How to Communicate with Zillions of Robots", **Proceedings of SPIE Mobile Robots VIII**, Boston MA, 9-10 September 1993, Volume 2058, pp 250-257.

6. Gage, D.W. "Cost-optimization of many-robot systems", **Proceedings of SPIE Mobile Robots IX**, Boston MA, 2-4 November 1994, Volume 2352, pp 260-266.

7. Everett, H.R., Gage, D.W., Gilbreath, G.A., Laird, R.T., and Smurlo, R.P. "Real-world issues in warehouse navigation", **Proceedings of SPIE Mobile Robots IX**, Boston MA, 2-4 November 1994, Volume 2352, pp 249-259.

8. Gage, D.W. "Security Considerations for Autonomous Robots", **Proceedings of Symposium on Security and Privacy**, Oakland CA, 22-24 April 1985, pp 224-228. Reprinted in **Computer Security Journal**, vol 6, 1990, pp 95-99.

9. Tanenbaum, A.S. **Computer Networks**. Prentice-Hall, Englewood Cliffs NJ, 1981, pp 253-259.

10. Gunawan, D.S., Pister, K.S.J., and Lin, L-Y. "Micromachined Corner Cube Reflectors as a Communication Link", **Sensors and Actuators A (Physical)**, Volume A47, number 1-3, March-April 1995, pp 580-583.

11. Wang, J. "On Sign-board Based Inter-robot Communication in Distributed Robotic Systems", **Proceedings of the 1994 IEEE International Conference on Robotics and Automation**, San Diego CA, May 1994, pp 1044-1050.

12. Arkin, R.C., Balch, T., and Nitz, E. "Communication of Behavioral State in Multi-agent Retrieval Tasks", **Proceedings of the 1993 IEEE International Conference on Robotics and Automation**, Atlanta GA, May 1993, pp III-588-594.

13. Lucarini, G., Varoli, M., Cerutti, R., and Sandini, G. "Cellular Robotics: Simulation and HW Implementation", **Proceedings of the 1993 IEEE International Conference on Robotics and Automation**, Atlanta GA, May 1993, pp III-846-852.

14. Winston, P.H, and Horn, B.K.P. **LISP Second Edition**, Addison-Wesley, Reading MA, 1984.

15. Hoskins, D.A. "A Least Action Approach to Collective Behavior", **Proceedings of SPIE Microrobotics and Micromechanical Systems**, Philadelphia PA, 25 October 1995, Volume 2593 (these proceedings).

16. Braitenberg, V. **Vehicles: Experiments in Synthetic Psychology**, The MIT Press/Bradford Books, Cambridge MA, 1984.

17. Mataric, M.J. "Issues and Approaches in the Design of Collective Autonomous Agents". To appear in **Robotics and Autonomous Systems**, 1995.